Greedy Algorithms

- Optimization problems
  - Dynamic programming, but overkill sometime.
  - Greedy algorithm:
    - Being greedy for local optimization with the hope it will lead to a global optimal solution, not always, but in many situations, it works.

An Activity-Selection Problem

- Suppose A set of activities S=$\{a_1, a_2, \ldots, a_n\}$
  - They use resources, such as lecture hall, one lecture at a time
  - Each a_i, has a start time s_i, and finish time f_i, with $0 \leq s_i < f_i < \infty$.
  - a_i and a_j are compatible if $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap
- Goal: select maximum-size subset of mutually compatible activities.
- Start from dynamic programming, then greedy algorithm, see the relation between the two.

DP solution –step 1

- Optimal substructure of activity-selection problem.
  - Furthermore, assume that $f_1 \leq \ldots \leq f_n$.
  - Define $S_{ij}=\{a_k: f_i \leq s_k < f_k \leq s_j\}$, i.e., all activities starting after a_i finished and ending before a_j begins.
  - Define two fictitious activities a_0 with $f_0=0$ and a_{n+1} with $s_{n+1}=\infty$
    - So $f_0 \leq f_1 \leq \ldots \leq f_{n+1}$.
  - Then an optimal solution including a_k to S_{ij} contains within it the optimal solution to S_{ik} and S_{kj}.

DP solution –step 2

- A recursive solution
- Assume c[n+1,n+1] with c[i,j] is the number of activities in a maximum-size subset of mutually compatible activities in S_{ij}. So the solution is $c[0,n+1]=S_{0,n+1}$.
- $$C[i,j]= \begin{cases} 0 & \text{if } S_{ij}=\varnothing \\ \max\limits_{i<k<j \text{ and } a_k \in S_{ij}} \{c[i,k]+c[k,j]+1\} & \text{if } S_{ij}\neq\varnothing \end{cases}$$

Converting DP Solution to Greedy Solution

- Theorem 16.1: consider any nonempty subproblem S_{ij}, and let a_m be the activity in S_{ij} with earliest finish time: $f_m = \min\{f_k : a_k \in S_{ij}\}$, then

1. Activity a_m is used in some maximum-size subset of mutually compatible activities of S_{ij}.
2. The subproblem S_{im} is empty, so that choosing a_m leaves S_{mj} as the only one that may be nonempty.

- Proof of the theorem:

Top-Down Rather Than Bottom-Up

- To solve S_{ij}, choose a_m in S_{ij} with the earliest finish time, then solve S_{mj}, (S_{im} is empty)
- It is certain that optimal solution to S_{mj} is in optimal solution to S_{ij}.
- No need to solve S_{mj} ahead of S_{ij}.
- Subproblem pattern: $S_{i,n+1}$.

Optimal Solution Properties

- In DP, optimal solution depends:
  - How many subproblems to divide. (2 subproblems)
  - How many choices to determine which subproblem to use. (j-i-1 choices)
- However, the above theorem (16.1) reduces both significantly
  - One subproblem (the other is sure to be empty).
  - One choice, i.e., the one with earliest finish time in S_{ij}.
  - Moreover, top-down solving, rather than bottom-up in DP.
  - Pattern to the subproblems that we solve, $S_{m,n+1}$ from S_{ij}.
  - Pattern to the activities that we choose. The activity with earliest finish time.
  - With this local optimal, it is in fact the global optimal.

Elements of greedy strategy

- Determine the optimal substructure
- Develop the recursive solution
- Prove one of the optimal choices is the greedy choice yet safe
- Show that all but one of subproblems are empty after greedy choice
- Develop a recursive algorithm that implements the greedy strategy
- Convert the recursive algorithm to an iterative one.

Greedy vs. DP

- Knapsack problem
  - I1 (v1,w1), I2(v2,w2),…,In(vn,wn).
  - Given a weight W at most he can carry,
  - Find the items which maximize the values
- Fractional knapsack,
  - Greed algorithm, O(nlogn)
- 0/1 knapsack.
  - DP, O(nW).
  - Questions: 0/1 knapsack is an NP-complete problem, why O(nW) algorithm?

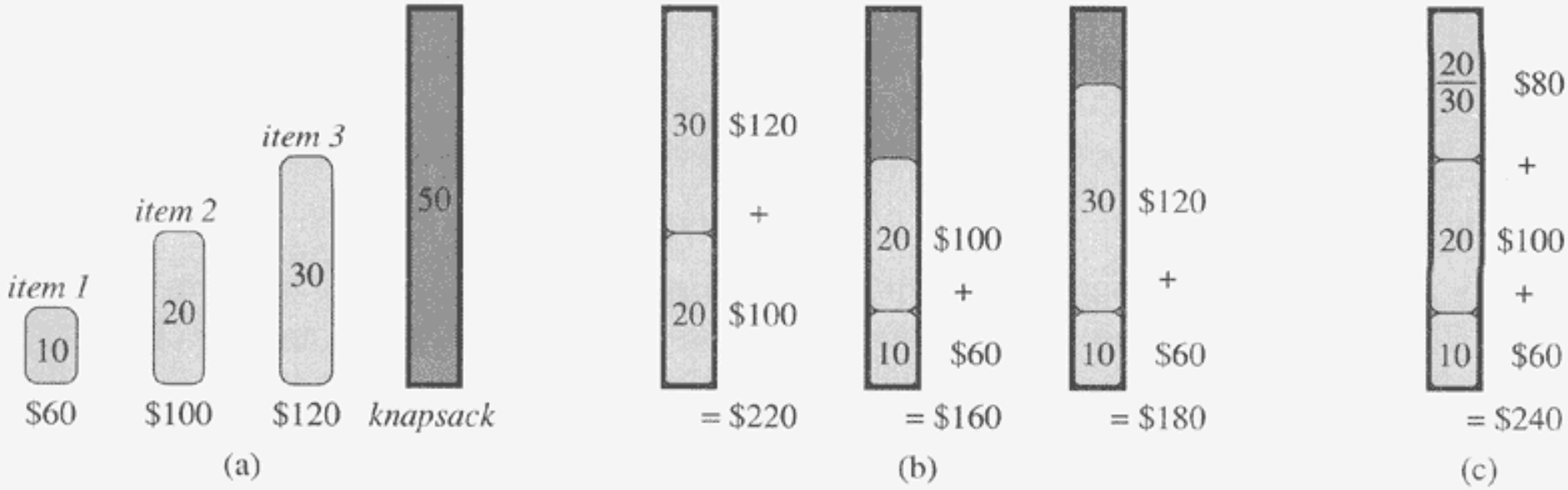

Figure 16.2 The greedy strategy does not work for the 0-1 knapsack problem. **(a)** The thief must select a subset of the three items shown whose weight must not exceed 50 pounds. **(b)** The optimal subset includes items 2 and 3. Any solution with item 1 is suboptimal, even though item 1 has the greatest value per pound. **(c)** For the fractional knapsack problem, taking the items in order of greatest value per pound yields an optimal solution.

Typical tradition problem with greedy solutions

- Coin changes
  - 25, 10, 5, 1
  - How about 7, 5, 1
- Minimum Spanning Tree
  - Prim's algorithm
    - Begin from any node, each time add a new node which is closest to the existing subtree.
  - Kruskal's algorithm
    - Sorting the edges by their weights
    - Each time, add the next edge which will not create cycle after added.
- Single source shortest pathes
  - Dijkstra's algorithm
- Huffman coding
- Optimal merge